Status of the Work to Parallelize ATLAS Reconstruction Processing version 1.0

Paolo Calafiura, Keith Jackson, Mous Tatarkhanov, Yushu Yao
Lawrence Berkeley National Lab

March 12, 2010

Abstract

Over the last six months we have tried to characterize the performance of athenaMP[1], the multiprocess version of athena, the ATLAS software framework, when running event reconstruction. Past athenaMP performance studies were performed on multicore platforms with low (2-4) core count. The initial focus of this work has been to extend these studies to transitional manycore platforms with higher (32) core count. Encouraged by the good scalability shown in these studies we are now investigating the causes of the departure from ideal scalability ($\sim$50% running on 32 cores). At the same time we have started detailed studies of ATLAS reconstruction on the Intel Nehalem platform which should be most common choice for the next round of hardware procurements.

1 Program of Work

In the June 2009 USLHC supplemental funding request we proposed the following deliverables for ATLAS (and CMS) developers:

1. ATLAS will use and maintain athenaMT/athenaMP and will concentrate on athenaMP possibly improving athenaMP's I/O buffer memory sharing.

2. While the new CMS event task farm is developed, we will start by benchmarking athenaMP, the multi-process version of the ATLAS athena framework which also implements an event task farm. LBNL NERSC is offering us access to 16-core and, later, 32-core Intel processors for our tests as well as support in setting up, running, and analyzing the results of IPM, NERSC parallel performance monitoring tool. ATLAS will also explore parallelization strategies that would allow to partition the data and or the work needed to reconstruct an event among multiple processing units (for example segmenting a detector in regions of interest, or pipelining the reconstruction tasks through a chain of multiple processing units.

3. Performance evaluations on N-core systems investigating application event throughput scaling, measuring the deviations from the ideal where event throughput per application instance remains constant as more processing units are added. The measurement will include cost factors for CPU, memory, and I/O.

4. Analysis of bottle necks at the point of non-ideal scaling, including suggestions for strategies to address them. The biggest challenge of this task will be to find (hopefully not develop) tools to measure memory bandwidth and to study how memory is shared between processing units.

5. Initiate investigations of "decorated algorithms" [1], selected based on profile results, decorate and measure the performance difference.

Over the course of these six months we have concentrated our efforts on the first three deliverables, and we have just started looking into the fourth.

2 athenaMP

athenaMP is an event-parallel version of athena, that runs serially through the reconstruction program configuration and initialization and then uses Unix fork to create a task farm of event workers. Thanks to Linux Copy On Write memory optimization each event worker process shares a large fraction (see Fig. 5) of its memory with the other workers and the mother process. At the end of the event loop, the mother process wait for all forked event worked to join back and then merges their output files.

The greatest advantage of athenaMP is that it offers a non-intrusive approach to parallelization. Each event worker runs isolated in its own address space. Aside from requesting the application to advertise which files are used for I/O, the physics code that runs serially is more or less guaranteed to run as is in athenaMP. Contrast this with the difficulties (described in [1]) experienced with ATLAS High-Level Trigger attempts to run athena event workers in multiple threads, and it should be apparent why athenaMP multi-process approach is the natural choice for ATLAS reconstruction on multi-core platforms.

3 Performance Measurements

3.1 Methodology

We run both full reconstruction, and fast reconstruction (a lightweight job processing only Inner Detector data) on a number of current and next generation hardware platforms. Unless otherwise specified the results in this note refer to full reconstruction and do not differ significantly between full and fast reconstruction.

[1]in the proposal we defined decorated algorithm as enhancements to existing loops and structures to add fine-grained parallelism using modern programming techniques

Table 1: Comparison of Computers

Name	**CPU**	**Cache**	**Arch**	**Technologies**	**Number of CPUs**	**Total Number of Cores**	**Total Mem**
Turing	AMD Opteron 8384	L1: 128 KB per Core L2: 512 KB per Core L3: 6 MB per CPU	P2P	HyperTransport	8	32	256 GB
Coors	Intel Xeon X5550	L1: 32 KB Data + 32 KB Inst. per Core L2: 256 KB per Core L3: 8 MB per CPU	P2P	HyperThreading, QuickPath, TurboBoost	2	8	24 GB
VoAtlas	Intel Xeon E5410	L1: 32 KB Data + 32 KB Inst. per Core L2: 8MB per CPU	Bus		2	8	16 GB

To improve the reliability of our results we have captured our entire measurement procedure in a python script that, for a given configuration, runs a series of jobs with increasing multiplicity, collects the results, and produces a set of standard performance histograms providing basic measurements like event throughput or memory usage per process, as well as more sophisticated ones characterizing the amount and relative weight of I/O operations, or memory access. We are working toward integrating our script into the nightly ATLAS run-time testing framework [2] to achieve continuous performance monitoring in a production environment.

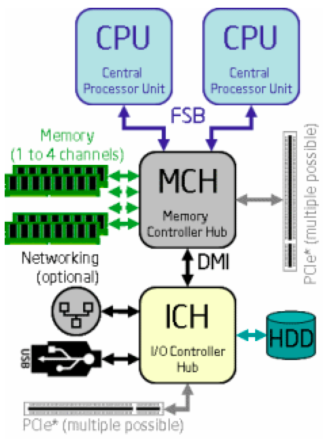
3.2 Hardware Used for Testing

Three test computers with 64-bit architecture are selected for our purposes[2]. The selection aimed to test the impact of different system architecture, CPU/memory access models on the performance of parallel ATLAS jobs.

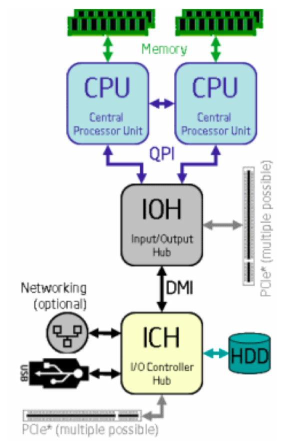
A comparison of the test computers is listed in Table.1. The VoATLAS computer has two Quad-Core Intel Xeon E5410 and is based on the Bus architecture. As shown in Fig.1 (a), each CPU is connected to the Memory Controller Hub (MCH) via a separate Front Side Bus (FSB), and the memory is attached to the MCH. In this architecture all memory access operations issued by the CPUs are processed by the MCH, so logically all CPUs have the same "distance" to any memory module. The MCH need to communicate with all CPUs via FSBs, the memory modules, and the PCI devices. This means the scalability of this architecture is limited due to speed limitations like the communication protocol and physical limitations like number of pins. It is unlikely the Bus architecture will support more than 8 modern physical CPUs.

The shared bus protocol, since it is shared by multiple devices, requires that a

[2]The Operating Systems are 64-bit Linux 2.6 (Scientific Linux 5 on Turing and VoAtlas and Ubuntu 9 on Coors), and ATLAS software is running in 32-bit compatibility mode.



(a)



(b)

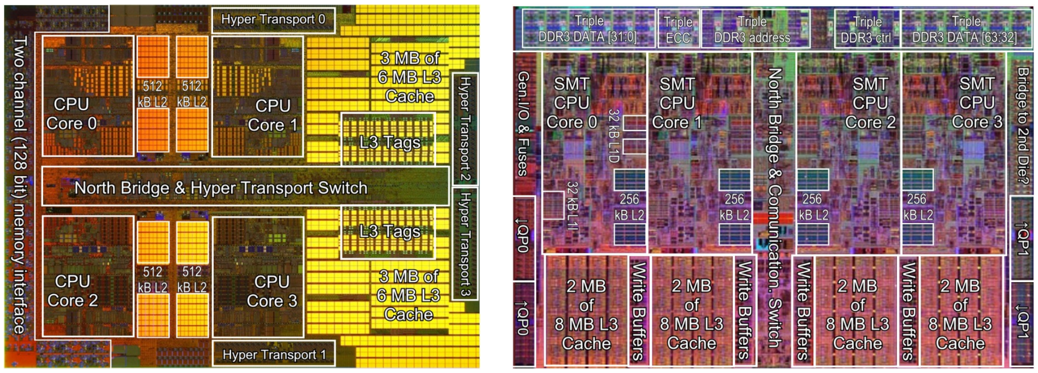
Figure 1: (a). Block diagram for the Intel Front Side Bus based architecture [5]; (b). Block diagram for the Intel QuickPath Architecture [4]

device on the bus must first arbitrate for control before sending a request. The Point-to-Point (P2P) architecture, on the contrary, mandates that there are only two devices, so request can be sent without any preliminary handshaking, hence greatly increase the communication bandwidth. AMD implemented this technology under the name "HyperTransport".

One example is the AMD Opteron 8384 Processor (codename "Shanghai") shown in Fig.2 (a). The quad-core processor includes 4 HyperTransport ports that can be connected to other processors or devices via the P2P protocol. It also includes a memory interface that will connect to memory modules.

National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (LBNL) provides us a SUN SunFire X4600 server (named "Turing") which is based on this architecture. Fig.3 shows the block-level diagram of the modules. Each quad-core processor is attached with 32 GB of DDR3 memory, this eliminates the need of a central Memory Control unit. All processors are inter-connected via HyperTransport, so that each processor is at most 3 steps away from any other processor (e.g. the distance between CPU0 and CPU2 is 1, the distance between CPU0 and CPU7 is 3, CPU0-CPU2-CPU7). Since memory modules are distributed over multiple processors (the so called non-uniform memory architecture, NUMA), accessing memory located in other processors will be consume much more time than accessing local memory. Coordinating the process/threads with their memory access patterns is very important for the overall performance of the applications. Combined with tools like affinity we can efficiently study the effect of these patterns on the performance of ATLAS jobs.

Intel implements similar P2P technology under the name "QuickPath Interconnect" (QPI)[6]. The computing science division at LBNL provides us a test



(a) (b)

Figure 2: (a) Die shot and function description for AMD Opteron 8384 Processor (codename “Shanghai””);(b) Die shot and function description for Intel Xeon X5550 (codename “Nehalem”)

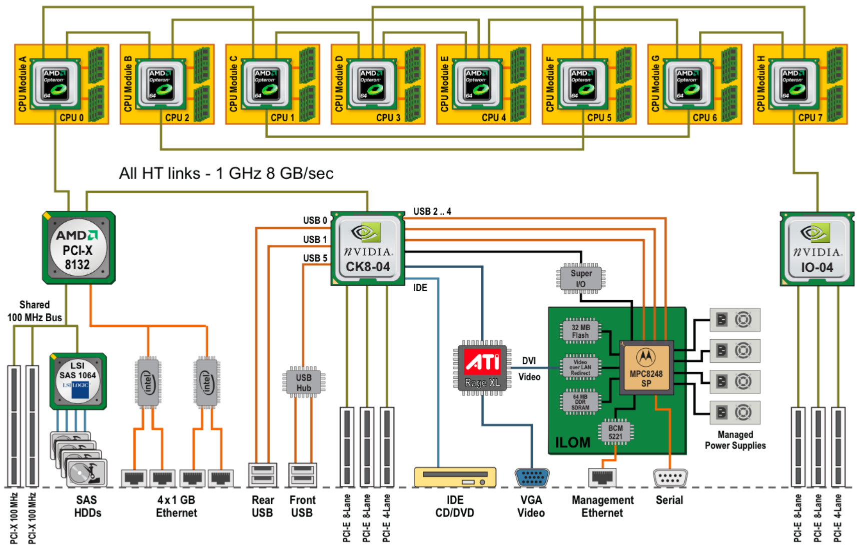


Figure 3: Sun Fire X4600 M2 motherboard block-level diagram[3]

computer (named “Coors”) based on this architecture. It consists of two quad-core Xeon X5550 processors (codename “Nehalem”, see Fig.2(b)). As shown in Fig.1(b), each processor is attached with 12 GB of memory, and the two processors are connected with QPI. Difference from AMD Shanghai, the Intel Nehalem includes the HyperThreading technology and the TurboBoost technology. Hyper-threading is used to improve the performance of parallelized jobs by presenting two virtual cores to the operating system for each physical core. This allows the operating system to schedule two threads/processes simultaneously to the same physical core. A hyper-threading equipped processor can efficiently share the execution pipeline of each core among its two hardware threads, hence increase the performance when running software with significant memory access latencies. The TurboBoost technology, on the other hand, allows CPU performance to be increased on demand by adjusting the clock frequency dynamically.

3.3 Tools

An important part of this project has been understanding how to measure the performance of athenaMP as it scales to more and more cores. It is essential to understand the details of athenaMP performance so that performance bottlenecks can be identified and removed. This has involved investigating, and using a wide variety of tools to measure a broad array of system and application parameters. Unfortunately, understanding performance measurements on modern multi-core systems is a difficult challenge. We have worked closely with several of the experts in this field at NERSC to help understand the data we have collected for athenaMP.

One useful tool for measuring performance is the linux sar command. The sar command can produce a wide variety of system activity reports based on the data collected by the system activity data collector (sadc) process. With sar we were able to measure the number of reads and writes from the physical disks, as well as the amount of data read. The sar command was also useful reporting the total system load and memory usage. We used another standard linux tool, vmstat, for examining the virtual memory system in more detail. The vmstat command gives information on the amount of virtual memory used, the amount of free memory, and the system paging activity. This information, when combined with the information available from the linux top command, allowed us to understand the large-scale memory behavior of athenaMP.

While the standard linux system tools provided a great deal of information, most of it was at the system level, and not the application level. To address this shortcoming, and try to correlate the application level behavior we were seeing with the system level statistics, we used the NERSC Integrated Performance Monitoring (IPM) tool. IPM was originally designed as a performance monitoring tool for parallel MPI applications. We worked closely with the IPM developers to help extend IPM to work with forked serial jobs such as athenaMP. IPM works by using the `LD_PRELOAD` mechanisms to interpose itself between the application and the standard system calls. In this way IPM can collect detailed information about the performance of the application. By correlating the IPM data with the system level data collected, we were able to begin to understand the behavior of athenaMP on multi-core.

With the understanding gained in our testing, we were able to focus on memory performance as a likely scaling bottleneck for athenaMP. Thus we have begun to examine how to measure memory usage and performance on NUMA architectures like that of the Intel Nehelam chips that are becoming widely prevalent. Modern linux kernels, if compiled correctly, provide a great deal of information about NUMA statistics. For each process, the `/proc` filesystem will contain a `numa_maps` file that contains information on what memory pages are located local to the process and which are located remotely. To obtain a more dynamic view of the same data, we use the numastat command. This command allows us to see the location of memory allocations as athenaMP makes them. Our initial tests indicate, as we expected, that the default NUMA policies in the linux kernel are not suitable for HPC applications like athenaMP. By default, the kernel does not always allocate memory close to the running process. Instead, a round-robin algorithm is used to distribute memory allocations over all of the memory locations. For most general-purpose applications, this is quite useful because it prevents a single process from using all of the memory in one location, thus starving another process that might run on the same core. For HPC applications, this policy may hinder performance. To address this, we are exploring the ability to set processor affinity to bind a process to a specific core, and using the numactl command to set NUMA policy for a process. With numactl we can force the linux kernel to allocate memory on the locations closest to the running process.

To confirm our hypothesis about NUMA performance, we would like to be able to measure the actual usage and bandwidth of the Nehalem QuickPath Interconnect. To do this, we are examining the usage of PAPI counters. PAPI provides an API to access the underlying hardware counters on the chip. Unfortunately, understanding the information returned is difficult. The counters give you numbers for accesses across the QuickPath Interconnect, but they include all system accesses. Working closely with researchers at NERSC, we are investigating exactly how these numbers are generated. For example, does a dirty cache line generate a message to all memory locations? It is still an open question as to whether PAPI will provide the information we need on detailed hardware memory accesses, or we will need to find other ways of measuring this.

3.4 Early Results

The main advantage of an event-parallel approach a la athenaMP compared to the job-parallel approach currently used in production, is that athenaMP event workers share a hopefully large amount of memory. Sharing memory between workers will allow either to reduce the amount of memory required for a "standard" ATLAS production node, or, alternatively, to increase the memory footprint of the reconstruction software we run in production.

Comparing the amount of real memory(Fig. 4) used by a serial job and by athenaMP jobs one sees that indeed the memory footprint of an event worker is about 50% smaller than that of a serial job.

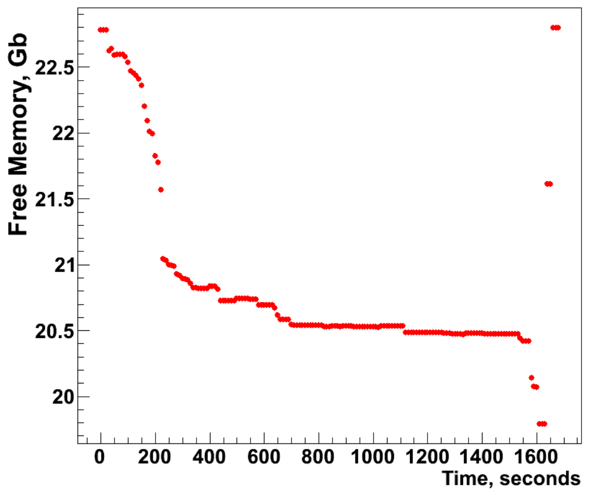


Figure 4: The amount of free memory in a node as a function of time. We consider the sudden increase in free memory observed at the end of a job as the amount of *real memory* used by the job

Having established that athenaMP brings a significant benefit in reducing the memory footprint, we need to measure how well athenaMP scales with the number of event workers running, in other words what is the cost in terms of CPU efficiency associated with athenaMP. As can be seen from Fig 6 athenaMP scales reasonably well up to 32 event workers on the AMD box (turing.nersc.gov), with a predictable slow degradation of performance of about 25% in the 2-32 workers range. What is more surprising is the immediate ∼25% drop in efficiency observed in the case of the Nehalem box (coors.lbl.gov).

We have investigated three possible sources for the departures from perfect scalability: conflicts in I/O access; suboptimal scheduling from the kernel, leading excessive context switches; and the related issues of non-uniform memory access (NUMA) and locality of memory page accesses.

We quickly convinced ourselves that I/O is not a factor for the reconstruction jobs we run. Both sar and IPM agree that the time spent in I/O is below 2%. Also, thanks to the large amount of RAM available on turing and tesla, we were able to run our jobs putting both input and output files in memory (using `dev/shm`), and at the other extreme on a network mounted disk and observed no significant difference in wall-clock time in the two cases. This was probably to be expected since the total I/O bandwidth required by an athena reconstruction job is less then 0.5MB/s. In the future we want to test how athenaMP performs in

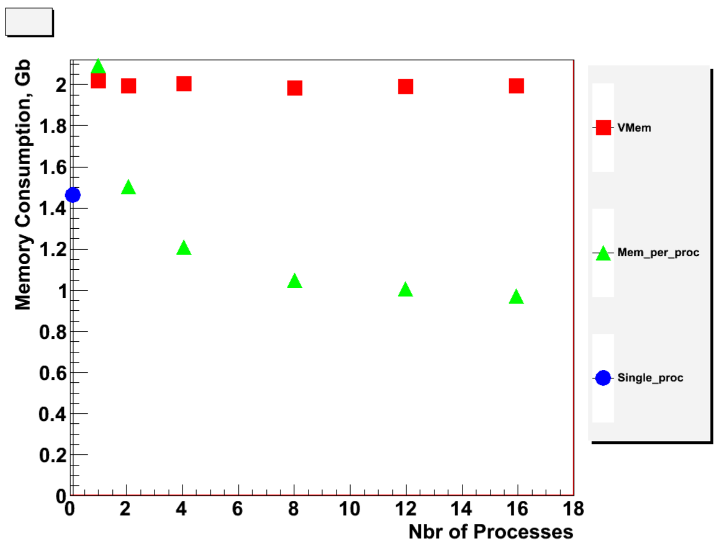


Figure 5: The amount of real memory used per event worker, as a function of the number of event workers. The first data point (blue circle, 0 processes) refers to a serial job.

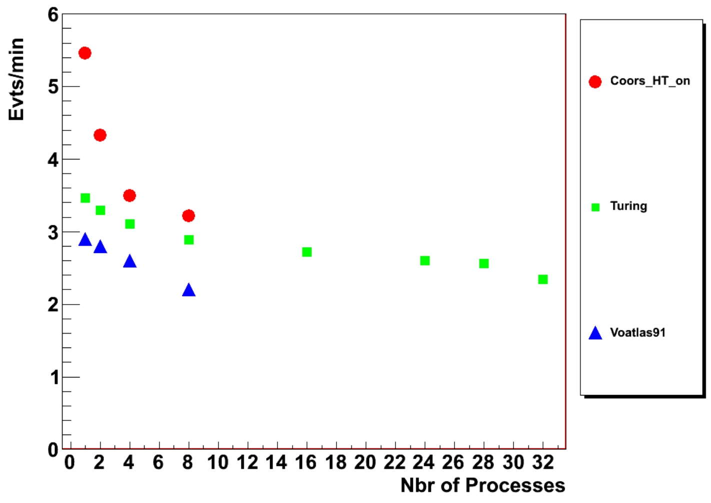


Figure 6: Events processed per minute of wall-clock time, per event worker, as a function of the number of event workers on voatlas91.cern.ch, coors.lbl.gov, and turing.nersc.gov. For a perfectly scalable application the event rate per worker should be flat.

analysis jobs, which use more than one order of magnitude extra I/O bandwidth.

Watching with `top` an athenaMP application running one sees clearly that the event workers keep moving from one core to another every few seconds. Naively this looks like an inefficient way to manage event workers which are running at 100% CPU. Every time an event worker is moved to another core the process will be stopped for hundreds (or thousands) of clock cycles and, after the switch, all cache lines and memory pages which where local for the process need to be transferred over to the new core. Recent linux kernels are reported to be pretty sophisticated in deciding on which core a certain process should run, as they attempt to match processes to locally accessible memory pages. In any case to test the effect of process scheduling and migration on athenaMP performance we used the linux `taskset` command, that allows to specify on which processing unit (physical or virtual) a certain process should run. Assigning processes to processing units in a well-balanced way[3], we were able to get rid of the sudden 25% drop in efficiency observed on coors (Fig. 7). What's more, when setting process affinity, we observe,anedoctically at this stage, that the event throughtput of multiple runs of the same job is more consistent than the one of free "floating" processes.

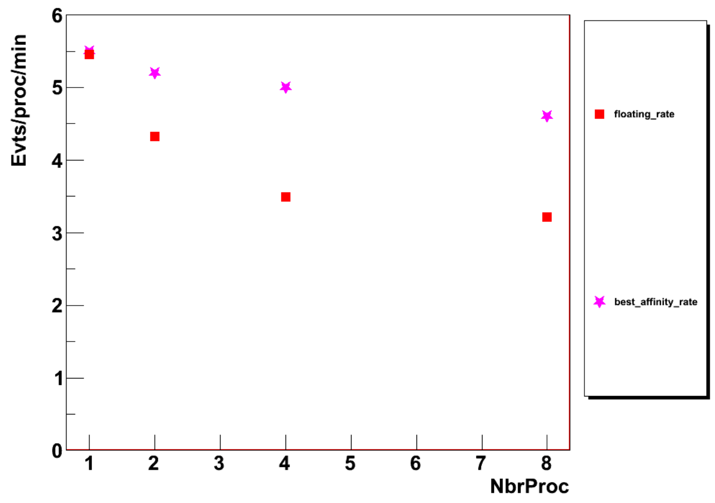


Figure 7: The effect of affinity on Nehalem (coors.lbl.gov), shown plotting the event rate per worker as a function of the number of athenaMP processes for balanced affinity settings and with "floating" processes scheduled by the kernel.

[3]we tried several balancing schemes and the results differs little for all reasonable schemes.

3.5 Current Work

Having established the beneficial effect of controlling process placement using affinity, we want to understand the residual ∼25% drop in efficiency observed when increasing the number of worker processes. As discussed in section 3.3 we are convinced that the limiting factor is the non-uniform access to memory. We are currently investigating usage of Linux `numactl` command to control the NUMA policy, and we are developing tools to analyze `/proc/pid/numa_maps` and the processor performance counters to obtain information about NUMA performance.

Another promising feature of the Nehalem architecture is Intel Hyperthreading support, the ability to address two virtual processing unit per physical core and share the work among them. Earlier work at CERN[7] indicated that overcommitting a Nehalem processor by 50% (submitting 12 jobs on a 8-core CPU) allowed to increase the processing rate by ∼20%. These studies were done using some generic benchmark, and we repeated them running ATLAS reconstruction using both serial athena and athenaMP on coors.lbl.gov. The results for job-parallellism are very encouraging(Fig. 8) and basically confirm CERN findings.

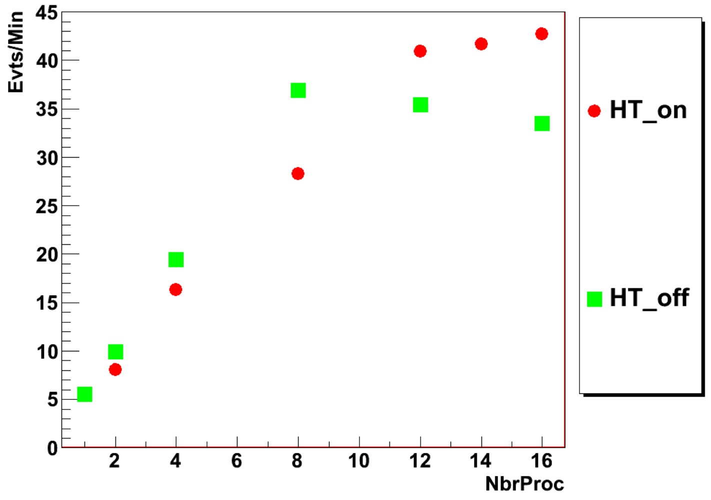


Figure 8: Average number of events processed per minute of wall-clock time, as a function of the number of event workers with Intel Hyperthreading support on or off, when running N separate athena instances (job-parallel athena).

Running athenaMP on coors with Hyperthreading support, shows even better performance improvements and scalability (Fig. 9). While this is a result that needs to be understood better, it may have a simple explanation: the total real memory footprint of an athenaMP job with 16 event processors is ∼16GB, while the total real memory footprint of the 16 separate jobs run by the job-parallel script is ∼24GB, very close to the 24GB of RAM installed on coors.lbl.gov, the Nehalem computer where these tests were run. While we are very pleased with this result, we have also observed that when running fast reconstruction jobs[4], athenaMP does not seem to profit much at all from Hyperthreading and only achieves ∼85% of the event throughput of the job-parallel runs (Fig. 10).

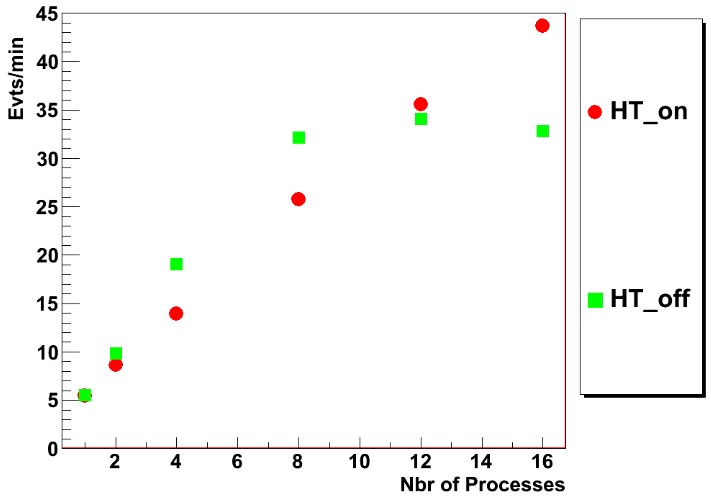


Figure 9: Average number of events processed per minute of wall-clock time, as a function of the number of event workers with Intel Hyperthreading support on or off, when running N event workers in athenaMP running full reconstruction.

4 Future Work

The main focus of our work for the remainder of FY10 will be to understand if and how NUMA limits the peak performance of athenaMP, and why athenaMP does not always seem to be able to benefit of Intel Hyperthreading support. We are trying to setup a collaboration with NERSC, UC Berkeley Parlab, and directly with Intel to characterize memory access performance in a reliable and reasonably simple way. Based on our initial investigations and discussions with experts at the aforementioned labs, we believe that tools to investigate memory performance in NUMA architectures are still in nascent form, and that it will need a significant amount of effort to define metrics that will allow ATLAS developers to identify the code introducing memory bottlenecks.

Related to this work we need to understand the impact of linux kernel optimizations, such us different NUMA or scheduling policies, on athenaMP performance.

[4]which have a memory footprint of ∼500MB, about one third of a full reconstruction job

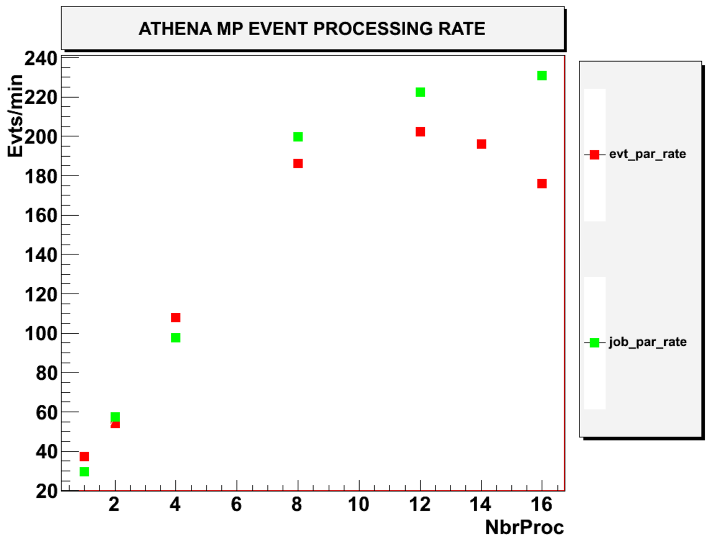


Figure 10: Effect of Intel Hyperthreading on job-parallel athena (green) and athenaMP (red) when running fast reconstruction. Compare to Figs. 8 and 9.

Given the disappointing event throughput of athenaMP running fast reconstruction, we want to start measuring athenaMP performance for typical AOD analysis jobs, which are even more lightweight than fast reconstruction, and which potentially may run into I/O bottlenecks.

Finally we want to investigate some architectural changes that could improve athenaMP scalability:

- an event server process, that by organizing disk reads in sequential reads of large block could improve event input speed. This would also allow to transform athenaMP in a sort of "event processing daemon" continously processing events while they become available and removing to a large extent the serial processing at job initialization and finalization, which for lightweight analysis jobs can easily add up to 50% of the job
- simmetrically we want to try to replace the current merging of output files at the end of the event loop with an event sink process[5], which besides centralizing and possibly speeding up disk writes, will make them asynchronous, removing the barrier at the end of the job, thereby reducing the total time to complete an athenaMP job.
- with an eye to massively many-core processors (128 cores and above) and with the goal of improving locality of memory access, reducing NUMA

[5]or a ROOT native mechanism for parallel output

problems, we want to start exploring techniques for sub-event parallelization. One such technique that we believe helds particular promise is breaking up of the athena control flow in multiple execution sequences minimally coupled in terms of data flow. We will then need a mechanism to efficiently let data flow between different subprocesses. One such mechanism would exploit athena Transient Data Store and the related Transient/Persistent separation[8] to exchange data between processes either via sockets, or via shared memory segments.

5 Acknowledgements

We gratefully acknowledge the contributions to this work from Pere Mato and Vincenzo Innocente (CERN), Sebastien Binet (LAL), Charles Leggett(LBNL), David Skinner, Karl Fuerlinger, and Nick Wright (NERSC).

References

[1] Harnessing multicores: strategies and implementations in ATLAS
`https://twiki.cern.ch/twiki/pub/Atlas/AthenaMP/athena-mp.pdf`

[2] The ATLAS RunTimeTester software
`http://indico.cern.ch/contributionDisplay.py?contribId=140&sessionId=60&confId=35523`

[3] Sun Fire X4600 M2 Server Architecture whitepaper
`http://www.sun.com/servers/x64/x4600/arch-wp.pdf`

[4] `http://ark.intel.com/Product.aspx?id=37106`

[5] `http://ark.intel.com/Product.aspx?id=28032`

[6] `http://www.intel.com/technology/quickpath/introduction.pdf`

[7] `https://twiki.cern.ch/twiki/bin/view/FIOgroup/ProcRefHyperthreading`

[8] The StoreGate: a Data Model for the Atlas Software Architecture
`http://www.slac.stanford.edu/econf/C0303241/proc/papers/MOJT008.PDF`